An optimisation of Gaussian mixture models for integer processing units

Real-Time Image Processing manuscript No.
(will be inserted by the editor)

Claudio Salvadori · Matteo Petracca · Jesus Martinez del Rincon · Sergio A Velastin · Dimitrios Makris

An Optimisation of Gaussian Mixture Models for integer processing units

Received: date / Revised: date

Abstract This paper investigates sub-integer implementations of the adaptive Gaussian mixture model (GMM) for background/foreground segmentation to allow the deployment of the method on low cost/low power processors that lack Floating Point Unit (FPU). We propose two novel integer computer arithmetic techniques to update Gaussian parameters. Specifically, the mean value and the variance of each Gaussian are updated by a redefined and generalised "round" operation that emulates the original updating rules for a large set of learning rates. Weights are represented by counters that are updated following stochastic rules to allow a wider range of learning rates and the weight trend is approximated by a line or a staircase. We demonstrate that the memory footprint and computational cost of GMM are significantly reduced, without significantly affecting the performance of background/foreground segmentation.

Claudio Salvadori
TeCIP Institute, Scuola Superiore Sant'Anna, via Moruzzi 1, 56124 Pisa Italy
Tel.: +39-050-5492037
E-mail: claudio.salvadori@sssup.it

Matteo Petracca
National Inter-University Consortium for Telecommunications, Pisa, Italy
E-mail: matteo.petracca@cnit.it

Jesus Martinez del Rincon
The Institute of Electronics, Communications and Information Technology (ECIT), Queens University of Belfast, BT3 9DT, UK
E-mail: j.martinez-del-rincon@qub.ac.uk

Sergio A Velastin
Department of Informatic Engineering, Universidad de Santiago de Chile
E-mail: sergio.velastin@ieee.org

Dimitrios Makris
Digital Imaging Research Centre, Kingston University, London, United Kingdom
E-mail: d.makris@kingston.ac.uk

1 Introduction

Background modelling (Piccardi (2004), Cheung and Kamath (2004), Radke et al (2005)) is a basic task for many computer vision applications, such as surveillance, road traffic monitoring, assisted living, etc. Normally, background modelling is based on pixel-wise operations that lead to both high computational cost and high memory requirements. Hardware parallelisation approaches may significantly speed up the processing, however they tend to have relatively high computational cost and high power consumption.

In Smart-Camera Network applications (Pagano et al (2012)), the goal is the pervasivity of the low cost visual sensors. The usage of very simple embedded systems based on low memory, low computational capability and low power consumption microcontrollers/microprocessors is required to both reduce costs and increase node power autonomy. Thus, microcontrollers without Floating Point Unit (FPU) are attractive, as FPUs increase the cost and the power consumption of the system. Consequently, one of the challenges is the redefinition of complex computer vision algorithms in an optimised and/or approximated version while maintaining comparable performance.

In this paper we present an extended version of what we proposed earlier in Salvadori et al (2012): we take the work further by considering any mixture of Gaussians (previously we only considered a mixture of two Gaussians), thus generalising the method to a generic mixture of G Gaussians. We also consider a larger domain for the learning-rate to cover a wider range of applications. Here we also propose two methods to approximate the Gaussian Mixture Model (GMM): iGMM-l and iGMM-s, where the integer weight quantisation is performed by a linear or a staircase function, respectively. Thus, while our previous works could be consider as a particular case of iGMM-l, because of the common usage of linear quantisation, in this paper we present the new method iGMM-s able to better the accuracy in weight approximation and to improve the overall segmentation performance.

In what follows we will use the term iGMM-x to generally refer to both variations above.

The rest of the paper is organised as follow: in Sec. 2 background model techniques in the literature are reviewed, focusing on solutions suitable for embedded system deployment; in Sec. 3 the proposed methodology is explained; in Sec. 4 the implementation details are depicted to introduce Sec. 5 where the performance evaluation of the considered techniques is presented and analysed.

2 State of the art

One popular approach for background modeling in embedded systems is the temporal median filter (see Lo and Velastin (2001), Cucchiara et al (2003)) and its approximations. For example, in Hung et al (2010), a *selection-based median filter* algorithm is proposed to reduce the computation and consequently to speed-up processing, and in McFarlane and Schofield (1995) a very fast and low memory footprint *approximated median filter* is depicted. In Rahimi et al (2005), another low computation and low memory footprint method is based on *running average*: each background pixel is updated by its difference with the corresponding value in the current image, using a value $\alpha \in [0, 1]$ called *learning rate*. Moreover, in Iannizzotto et al (2010), a combination of the two previous methods is described to increase the robustness of the *median filter* to sudden changes.

The above mentioned techniques represent the variations of each image pixel using only one value to model the background, while foreground is considered as an outlier. Methods that explicitly represent foreground as part of *multimodal approaches* have demonstrated higher accuracy. For instance, in Apewokin et al (2009) a *multimodal mean technique* is presented: every image pixel is described by a group of K mean values and a fixed threshold, to define a membership criterion, and a pixel is labeled as background if it matches a popular mean value. However, all above assumptions lack a rigorous statistical base, as no deviation measurement is used to adapt the membership threshold.

Gaussian Mixture Method (see Stauffer and Grimson (1999)) explicitly provides rigorous statistical models for both background and foreground by representing each pixel by a set of adaptive G Gaussians. Although it allows performing solid and accurate background subtraction, this method is characterized by a high memory footprint, as each Gaussian is represented by three floating point parameters, and a high computation cost, as each parameter is updated using a Finite Impulse Response (or FIR) filter.

There have been only few reported attempts to optimize the above mentioned approach for embedded systems. An efficient implementation of the GMM algorithm is described in Shen et al (2012). The image is segmented into 8x8 blocks, and their projections (based on *compressive sensing*, proposed by Candes et al (2006) and Donoho (2006)) are modelled as a GMM, so as to label each one as background or foreground. Finally, each foreground block is refined to generate a pixel-wise binary map. This approach obtains very good performance in terms of processing speed (five times faster than the original GMM) but only a modest result in terms of memory (a quarter of the original GMM).

Finally, in Salvadori et al (2012), an integer based technique is described to allow deployment on processors with no FPU. Mean value and variance updating process uses a rounding operator, while weight updating is based on a counter-based representation. This technique demonstrated both low computation cost and a highly optimised memory footprint (1/12 of the original GMM memory footprint) for a two-Gaussian mixture. For example, they achieved real-time performance for a QQ-VGA resolution using a low cost/low power microcontroller (PIC32-80MHz). However its extension to models with more than two Gaussians is not trivial and it introduces restrictive limitations on the learning rate domain. In this paper we propose how to overcome those limitations and deal with any given number of Gaussians. Particularly, in order to improve the mean value and the variance updating rule we have enhanced the generalised round operations making use of a variable *updating step* ξ as a function of the learning rate α ($\xi(\alpha)$, see Sec. 3.4), and not constant as in our previous work. Moreover, in Sec. 3.5.1, we propose a technique to overcome the limitations introduced by the weight updating rule based on stochastic updating: the weights are represented as counter (as in Salvadori et al (2012)), but they are updated with a probability-based function so as to better emulate the original GMM.

3 Methodology

3.1 Introduction to Gaussian Mixture Model (GMM)

GMM is a technique to model the static background and separate it from foreground in video sequences acquired by static cameras. The temporal variation of each pixel of the image is modeled by the weighted sum (or mixture) of G Gaussians, each one of them described by three parameters: *mean value* (μ), *variance* (σ^2) and *weight* (w). Each parameter takes values from a certain range as presented in Tab. 1. Particularly, constraining the range of variance prevents Gaussians from degenerating into a *Dirac delta* (if $\sigma^2 \to 0$) or into a *uniform distribution* (if $\sigma^2 \to \infty$).

μ	σ^2	w
[0, 255]	$[\sigma^2_{min}, \sigma^2_{MAX}]$	[0, 1]

Table 1: Gaussian parameters ranges

The GMM adaptation procedure is based on the computation of the *difference* (see Eq. 1) of the pixel $p_i(j)$ (where i represents the pixel position and j the given frame) from the mean value of each g Gaussian, $\forall g \in [1, G]$.

$$d_{i,g}(j) = p_i(j) - \mu_{i,g}(j) \tag{1}$$

A *membership criterion* (see Eq. 2) is defined to update the Gaussians with the corresponding pixel value, given a certain value of *learning rate* α ($\alpha \in [0, 1]$) and a certain threshold T. Thus, if a pixel satisfies the membership criterion for a certain Gaussian, its mean value, variance and weight are updated using three different FIR filters described respectively by equations Eq. 3, Eq. 4 and Eq. 5. Otherwise, the weight of the considered Gaussian, is updated using Eq. 6.

$$\text{if } d^2_{i,g}(j) < T\sigma^2_{i,g}(j) \tag{2}$$

$$\mu_{i,g}(j+1) = \mu_{i,g}(j) + k_{i,g}(j)d_{i,g}(j) \tag{3}$$

$$\sigma^2_{i,g}(j+1) = \sigma^2_{i,g}(j) + k_{i,g}(j)(d^2_{i,g}(j) - \sigma^2_{i,g}(j)) \tag{4}$$

$$w_{i,g}(j+1) = (1-\alpha) \cdot w_{i,g}(j) + \alpha \tag{5}$$

else

$$w_{i,g}(j+1) = (1-\alpha) \cdot w_{i,g}(j) \tag{6}$$

where $k_{i,g}(j) = \frac{\alpha}{w_{i,g}(j)}$.

For each pixel i and each frame j, the weights of Gaussians satisfy the following equation:

$$\sum_{g=1}^{G} w_{i,g}(j) = 1 \tag{7}$$

The role of the weight is to discriminate between foreground and background Gaussians. For every pixel, the Gaussians are ordered accordingly to a *sorting rule*, based on the value of ρ (Stauffer and Grimson (1999)) that is directly proportional to the squared value of the weight, as described in Eq. 8. If the pixel belongs to the g-th Gaussian, where g satisfies the equation $\sum_{m=1}^{g} w_m(j) \leq T_W$ (where T_W is a threshold given by the user), the pixel is labeled as background, otherwise as foreground.

$$\rho(j) = \frac{w^2(j)}{\sigma^2(j)} \tag{8}$$

3.2 Overview of iGMM-x

In the following sections we propose integer-GMM or iGMM, a method to port the GMM to architectures that lack a FPU, aiming to minimise both memory footprint and processing time. To this purpose, we use *sub-integer representation*, where the Gaussian parameters are represented using a number of bits less than the size of the smallest data integer type (namely *uint8_t*). Thus, the mean value and the variance updating that depend directly on the pixel values, will be handled defining a *generalised round* operation (see Sec. 3.3). On the other hand, weight updating, dependent only on matching a certain Gaussian, will be handled using a stochastic approach (see Sec. 3.5).

3.3 Mean value and variance updating

This section describes the proposed updating operations of the mean value and the variance of a Gaussian, represented by sub-integer variables. To permit this operation, we have an *updating methodology* based on three phases, as shown in Fig. 1: (i) the *scaling transform*, (ii) the *additive updating function*, and (iii) the *scaling inverse transform*.

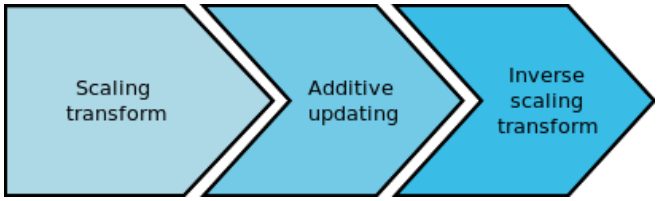


Fig. 1: Mean value and variance updating chain.

This updating methodology can be seen as a common pipeline used in several signal processing applications: data defined in the integer space $\mathbb{V}$ (in this case the domain represented in memory) are transformed to another space $\mathbb{U}$ (by using the scaling transform) where it is simpler to process, and then converted back to $\mathbb{V}$ (with scaling inverse transform). Moreover, the processing operation, performed in the GMM case in the real domain (namely $\mathbb{R}$), is ported in the $\mathbb{U}$ domain using a method based on the additive updating function and the generalised round. All the above mentioned phases are described in the following sections.

3.3.1 Sub-integer operations

To achieve sub-integer accuracy in Eq. 3 and Eq. 4, all their operands are converted appropriately using a scaling transformation. Accuracy is specified by a predefined parameter P and the conversion operation ($\chi(\circ) : \mathbb{V} \to \mathbb{U}$) for a given operand $V \in \mathbb{V}$ is defined as:

$$\chi(V) = V \cdot P \tag{9}$$

After the updating process, the resulting parameters of applying the operation in the new space are converted back to be saved in the memory (namely from $\mathbb{U}$ to $\mathbb{V}$), using the inverse transform shown in Eq. 10 ($(\chi^{-1}(\circ) : \mathbb{U} \to \mathbb{V})$).

$$\chi^{-1}(E) = \frac{E}{P} \tag{10}$$

with $E \in \mathbb{U}$ being the transformed operand. Both transformations in Eq. 9 and 10 are linear.

According to the above considerations, a *generalised-round* operation *G_ROUND* is defined (see also Salvadori et al (2012)) to enable a satisfactorily accurate representation of floating-point numbers in the integer domain $\mathbb{U}$, with $\mathbb{U} \subseteq \mathbb{Z}$. The above mentioned integer domain $\mathbb{U}$ can be described as the *discrete* and *ordered* set of numbers $\mathbf{B} = \{b_m \in \mathbb{U} | b_{m+1} - b_m = \gamma\}$, and it can be characterized by two parameters: the *granularity* γ, and the *updating step* ξ, defined as a function of γ:

$$\xi = \mathcal{F}(\gamma) = R\gamma \tag{11}$$

where $R \in (0, 1)$ is called *rounding parameter*. Therefore, it is possible to define the γ-floor operator $\lfloor \circ \rfloor_\gamma$ as:

$$\lfloor \delta \rfloor_\gamma = \lfloor \frac{\delta}{\gamma} \rfloor \gamma \tag{12}$$

and finally the *generalised-round* operator *G_ROUND* as:

$$G\_ROUND(\delta, \xi, \gamma) = \begin{cases} \lfloor \delta \rfloor_\gamma & \text{if } |\delta| \geq \gamma \\ \gamma & \text{if } (|\delta| < \gamma) \wedge (|\delta| \geq \xi) \\ 0 & \text{if } (|\delta| < \gamma) \wedge (|\delta| < \xi) \end{cases} \tag{13}$$

where $\delta \in \mathbb{Z}$ is the value to be rounded.

3.3.2 Generalised additive updating function

The second updating block in Fig. 1 corresponds to Eq. 3 and Eq. 4, which defines *additive updating operations* for both the mean value and the variance. However, since our proposed system is based on integer representation, (see Sec. 3.3.1), these formulas need to be redefined appropriately to emulate the original floating point formulation. Let $U : \mathbb{R} \to \mathbb{D}$ (in this case $\mathbb{D} \subseteq \mathbb{R}$) an *additive updating function* of the parameter a, such that $a(j+1) = U(a(j), \delta(j)) = a(j) + \delta(j)$, where $\delta(j)$ is the *updating contribution* for the parameter a. Therefore, Eq. 3 and Eq. 4 can be rewritten as:

$$\mu(j+1) = \mu(j) + \delta_\mu(d(j)) \quad \text{where } \delta_\mu(d(j)) = k(j)d(j) \tag{14}$$

$$\sigma^2(j+1) = \sigma^2(j) + \delta_{\sigma^2}(d^2(j)) \quad \text{where } \delta_{\sigma^2}(d^2(j)) = k(j)(d^2(j) - \sigma^2(j)) \tag{15}$$

According to the first block of Fig. 1, a conversion in the space $\mathbb{U}$ is needed:

$$\chi[a(j+1)] = \chi[a(j) + \delta(j)] = \chi[a(j)] + \chi[\delta(j)] \tag{16}$$

Consequently, to achieve the floating point accuracy with a sub-integer representation, the *generalised additive updating function* $\bar{U}$ is defined in the integer space $\mathbb{U}$ ($\bar{U}(\circ) : \mathbb{U} \to \mathbb{U}$), taking into account the *generalised-round* operator *G_ROUND* defined in Eq. 13, as follows:

$$\bar{U}(a(j), \delta(j)) = G\_ROUND[\chi[a(j+1)]] = \\ = \chi[a(j)] + \{G\_ROUND[\chi[\delta(j)]]\} \tag{17}$$

Finally, the overall result of all the chain in Fig. 1 can be summarized by the following relation:

$$a(j+1) = a(j) + \chi^{-1}\{G\_ROUND[\chi[\delta(j)]]\} \tag{18}$$

and the parameters γ and ξ for the *G_ROUND* operator should have appropriate values to produce similar results to the original updating function $U(a(j), \delta(j))$.

3.4 Calculating the updating step ξ

As described in our previous work Salvadori et al (2012), using a fixed value of rounding parameter R in Eq. 11 (in that case $R = 0.5$) limits the learning rate range, since updating the mean value and the variance is possible for only a sub-set of learning rate values. In this paper a more general technique is suggested. It computes the value of the updating step ξ as a function of the learning rate α, to ensure appropriate updating of both parameters while maintaining the membership intervals of Gaussians similar to the ones estimated by the floating-point approach.

According to Eq. 2, 14 and 15, both the updating contributions $\delta_\mu(d_{i,j})$ and $\delta_{\sigma^2}(d^2_{i,j})$ are bounded inside *updating contribution ranges*, as defined by the following formulas:

$$\delta^\mu_{min}(\sigma^2) = -k\sigma\sqrt{T} \leq \delta_\mu(d_{i,j}) \leq k\sigma\sqrt{T} = \delta^\mu_{MAX}(\sigma^2) \tag{19}$$

$$\delta^{\sigma^2}_{min}(\sigma^2) = -k\sigma^2 \leq \delta_{\sigma^2}(d^2_{i,j}) \leq k\sigma^2(T-1) = \delta^{\sigma^2}_{MAX}(\sigma^2) \tag{20}$$

where k depends on the learning rate α, as stated in Sec. 3.1.

Thus, fixing α, Eq. 19 and 20 can be seen as generic ranges bounded by a maximum (δ_{MAX}) and a minimum (δ_{min}), and they can be represented using a couple of parameters: the center C (see Eq. 21) and the radius r (see Eq. 22), as shown in Fig. 2.

$$C = \frac{\delta_{MAX} + \delta_{min}}{2} \tag{21}$$

$$r = \frac{\delta_{MAX} - \delta_{min}}{2} \tag{22}$$

In Tab. 2 both the intervals of Eq. 19 and 20 are represented in terms of centers and radii as a function of σ^2.

Because of the bounded nature of the variance and the direct proportionality of the above mentioned parameters on its value, it is possible to show that both the

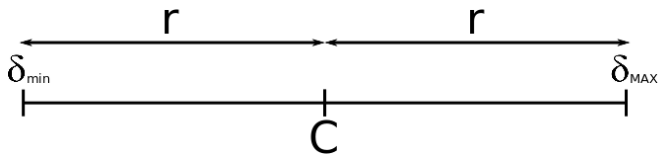


Fig. 2: Range center (C) and radius (r).

$C_\mu(\sigma) = 0$	$r_\mu(\sigma^2) = \alpha\sigma\sqrt{T}$
$C_{\sigma^2}(\sigma^2) = \dfrac{\alpha T\sigma^2 - 2\alpha\sigma^2}{2}$	$r_{\sigma^2}(\sigma^2) = \dfrac{\alpha\sigma^2 T}{2}$

Table 2: Centers and radii for mean value and variance updating contribution range.

centers and the radii are bounded into the following intervals:

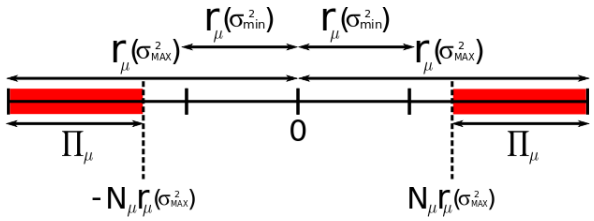
$$C_\mu = 0 \qquad \forall\sigma^2 \tag{23}$$
$$r_\mu \in [r_\mu(\sigma^2_{min}), r_\mu(\sigma^2_{MAX})] \tag{24}$$
$$C_{\sigma^2} \in [C_{\sigma^2}(\sigma^2_{min}), C_{\sigma^2}(\sigma^2_{MAX})] \tag{25}$$
$$r_{\sigma^2} \in [r_{\sigma^2}(\sigma^2_{min}), r_{\sigma^2}(\sigma^2_{MAX})] \tag{26}$$

From the previous considerations it is possible to state that the updating contribution ranges have a variable width, dependent on the variance. Consequently, the idea is to define *critical intervals* close to the border of both the above mentioned ranges, where the updating contribution is considered sufficiently large to increase the counter. In the following the criteria to estimate these intervals are described. By comparing Eq. 3 and 4, it is noted that the former depends directly on the difference value $d_{i,g}(j)$ (where $d_{i,g}(j) \in \mathbb{Z}$) but the latter evolves with its square value. Consequently, in case of a large value of $d_{i,g}(j)$, the variance update will be significantly higher than the mean value update, and then the above mentioned critical intervals can be estimated as:

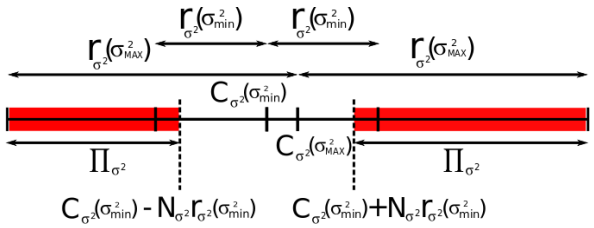
(a) the critical interval Π_μ is the zone inside the maximum width membership interval (namely the interval $[-r_\mu(\sigma^2_{MAX}), r_\mu(\sigma^2_{MAX})]$) where the updating contribution δ_μ is considered sufficiently large to update the mean value of its granularity;
(b) the critical interval Π_{σ^2} is the zone inside the minimum width membership interval (namely the interval $[C_{\sigma^2}(\sigma^2_{min}) - r_{\sigma^2}(\sigma^2_{min}), C_{\sigma^2}(\sigma^2_{min}) + r_{\sigma^2}(\sigma^2_{min})]$) where the updating contribution δ_{σ^2} is considered sufficiently large to update the variance of its granularity.

In Sec. 5.1.1 the impact of both of these principles will be shown and evaluated in a real test case.

Following the above considerations, two new parameters, *minimum updating numbers* $N_\mu \in \mathbb{R}$ and $N_{\sigma^2} \in \mathbb{R}$ (with $N_\mu \in (0,1)$ and $N_{\sigma^2} \in (0,1)$), can be used to code



(a) Mean value case.



(b) Variance case.

Fig. 3: Updating contributions ranges and *minimum updating number*.

the width of the critical intervals Π_μ and Π_{σ^2} respectively (see red zones in Fig. 3a and Fig. 3b). The introduction of these two parameters allows us to define the intervals while using only integers instead of real numbers for our calculations. Thus, according to Eq. 2, 3 and 4, the following two conditions describe these critical intervals:

$$\Pi_\mu = \{\delta_\mu \in \mathbb{Z} : |\delta_\mu - C_\mu| \geq \lfloor N_\mu r_\mu(\sigma^2_{MAX})\rfloor\} \tag{27}$$
$$\Pi_{\sigma^2} = \{\delta_{\sigma^2} \in \mathbb{Z} : |\delta_{\sigma^2} - C_{\sigma^2}| \geq \lfloor N_{\sigma^2} r_{\sigma^2}(\sigma^2_{min})\rfloor\} \tag{28}$$

Thus, it is possible to rewrite the generalised round operation (Eq. 13) as:

$$G\_ROUND(\delta,\xi,\gamma,C) = \begin{cases} \lfloor\delta\rfloor_\gamma & \text{if } |\delta| \geq \gamma \\ \gamma & \text{if } (|\delta| < \gamma) \wedge (\delta \in \Pi) \\ 0 & \text{if } (|\delta| < \gamma) \wedge (\delta \notin \Pi) \end{cases} \tag{29}$$

where ξ is the updating step, and Π is the critical interval. Thus, from Eq. 27, Eq. 28 and Eq. 29 it is possible to define the updating steps ξ as:

$$\xi_\mu(\alpha) = N_\mu(\alpha) \cdot r_\mu(\sigma^2_{MAX}, \alpha) \tag{30}$$
$$\xi_{\sigma^2}(\alpha) = N_{\sigma^2}(\alpha) \cdot r_{\sigma^2}(\sigma^2_{min}, \alpha) \tag{31}$$

Finally, the values of $N_\mu(\alpha)$ and $N_{\sigma^2}(\alpha)$ are calibrated iteratively to minimize the difference of the iGMMx and GMM membership intervals on a validation dataset, as depicted in Sec 5.1.1.

3.5 Weight updating

The weight updating rule, in contrast to the mean and variance updating rules, does not depend directly on the value of the pixel p_i, but only on the membership to a certain Gaussian: if the pixel belongs to a Gaussian, its weight is increased by a given value, otherwise it is decreased. In the original approach described in Sec. 3.1, the weights are represented as floating points between 0 and 1 and updated using the formulas in Eq. 5 and Eq. 6 to increase and decrease the weight respectively. To avoid the use of floating point weights, Eq. 5 and Eq. 6 are solved as a function of both the learning rate α and the number of iterations s needed to reach a certain value of weight w_g starting from 0 and 1 respectively. The derived equations have an exponential form as described in Eq. 32 and 33 (see also Fig. 4), where g is the considered Gaussian index ($g \in [1, G]$):

$$w_g(s) = 1 - (1 - \alpha)^s \tag{32}$$

$$w_g(s) = (1 - \alpha)^s. \tag{33}$$

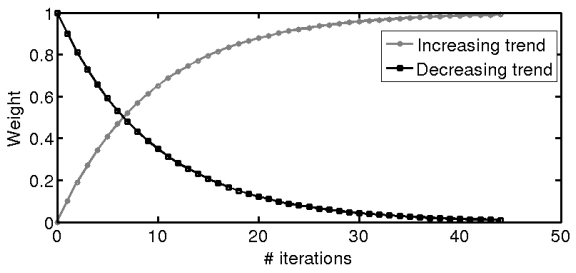


Fig. 4: Weight trends.

Under this formulation, solving Eq. 32 as a function of both the learning rate α and the weight, Eq. 34 is obtained: this relation specifies how many iterations $s(w, \alpha)$ are needed to reach a certain weight value starting from 0.

$$s(w, \alpha) = \frac{\log_{10}(1 - w)}{\log_{10}(1 - \alpha)} \tag{34}$$

Consequently, it is possible to represent a weight as an integer counter and represent it using a fixed number of l bits, able to cover the range of values $[0, L]$ (where $L = 2^l - 1$).

Though these considerations simplify the weight updating problem, the operation to retrieve the weight value continues to be floating point based because of the exponential nature of both Eq. 32 and Eq. 33. To deal with this issue two options are considered, where the weight trend is simplified and approximated to: (i) a line (see Fig. 5a and Sec. 3.5.2) or (ii) a staircase (see Fig. 5b and Sec. 3.5.3). These two approximation methods, also called *iGMM-l* and *iGMM-s* respectively, derive directly from Eq. 34 and they allow the computation of: (i) the number of steps needed to reach a certain value of weight starting from 0 (or 1); (ii) the number of steps needed to reach a certain value of weight from another one.

However, from Eq. 34, it is possible to show that the number of steps to retrieve the highest value of weight (namely $1 - \epsilon$ where ϵ is a very small positive number, e.g. $\epsilon = 0.01$) is inversely proportional to the learning rate, and consequently the lower the learning rate, the higher the number of bits needed to represent the weight. Because of the limited number of available bits, the deterministic calculation of both the linear and staircase trends could be efficient only for large values of α.

To overcome such restriction, we propose a stochastic method in Sec. 3.5.1 that increases the counter with a certain probability to reach the highest value of weight (namely $1 - \epsilon$) in a certain number of steps A in average using the above mentioned number of bits l, such that $A > L$ (where $L = 2^l - 1$).

3.5.1 Stochastic updating

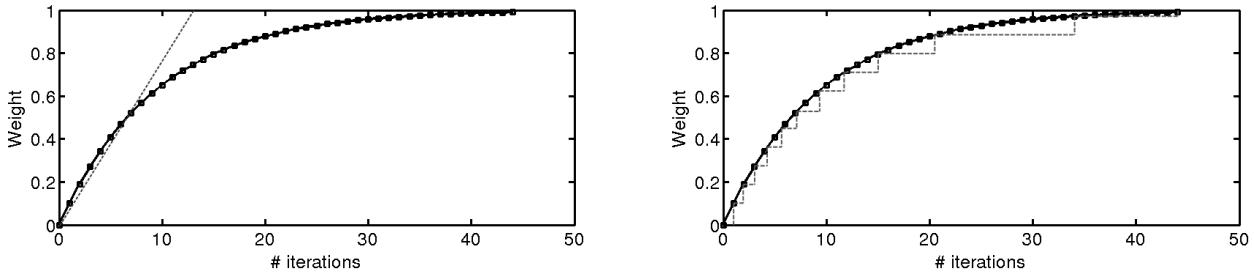
This section describes a method to generalise both the proposed weight approximations (i.e. iGMM-l and iGMM-s), to emulate floating point updating. Particularly, in the case of iGMM-l, it allows to overcome the learning rate limitation described in Salvadori et al (2012). In iGMM-s, stochastic updating is essential since it allows emulating a logarithmic trend using a set of weight values linearly sampled.

As stated in the previous section, the idea is to represent the weight using an integer counter and to retrieve its real value using one of the above mentioned approximations. Consequently, in an ideal scenario with no memory constraints, dealing with the weight updating means increasing the above mentioned counter by a certain value, called $STEP$. However, in case where weights are represented using l bits this approach is not feasible: reaching the highest value of weight when learning rate is relatively small may require A iterations, such that $A > L$ (where $L = 2^l - 1$). Thus a stochastic approach is proposed in order to overcome this limitation: the integer counter s is updated a $STEP$ amount with a certain probability to reach the highest value of weight in a certain A number of steps in average, as described in the following equation.

$$s_n = s_n + \mathbf{H}_n \tag{35}$$

where $\mathbf{H}_n \in \{0, STEP\}$ is a set of independent binary random numbers, such that:

$$E\left[\sum_{n=1}^{A} \mathbf{H}_n\right] = L \tag{36}$$



(a) iGMM-l. (b) iGMM-s.

Fig. 5: The weight trend and its approximations.

If the probability of the values $STEP$ and 0 are $P_{STEP}(n)$ and $P_0(n)$ respectively, from discrete random number theory and because of the linearity of the expected value $E[\circ]$, it is simple to obtain the following relation:

$$STEP \cdot \sum_{n=1}^{A} P_{STEP}(n) = L \tag{37}$$

The value of $P_{STEP}(n)$ is known and derives from the chosen weight approximation (see Sec. 3.5.2 and 3.5.3). The idea is to implement the logic derived from Eq. 35 making use of the set of the values $P_{STEP}(n)$, a uniformly distributed random number $\mathbf{X}_n$ defined in the interval $[0, MAX]$, and a threshold T_X, such that:

$$P_{STEP}(n) = P(\mathbf{X}_n > T_X) \tag{38}$$

It is possible to demonstrate (see Appendix A) that the threshold T_X is simply specified using the following formula:

$$T_X = [1 - P_{STEP}(n)] \cdot MAX \tag{39}$$

3.5.2 iGMM-l: Linear approximation

In this configuration, the relation to calculate the weight value w from the number of iterations s (Eq. (32)) is approximated by a line passing through the origin and the point $P_0 = (s(w_0, \alpha), w_0)$ as shown in Fig. 5a. This can be rewritten as the equation of the line that approximates the weight trend, depicted in Eq. 40.

$$w(s) = s \frac{w_0}{s(w_0, \alpha)} \tag{40}$$

Assuming the values of P_0 are known, the weight can be represented as a counter, and it is updated adding the value $STEP$ (usually equal to $G-1$). Consequently, the updating rules of using the linear approximation are the following: (i) the counter related with the g-th Gaussian has to increase by the updating step $STEP$; (ii) all the weights w_k (with $k \neq g$) have to decrease by the constant value $\frac{STEP}{G-1}$.

In this method it is important to estimate the optimal position of the point P_0, which can be calculated as the point where the line intersects the logarithmic trend of Eq. 32. This point represents an estimation where a weight swap may happen during the sorting operation. Because a precise estimation of the above mentioned swap point is impossible (the sorting depends on ρ, see Eq. 8), the considered value w_0 is the middle point of the weight range, namely 0.5.

As shown in Salvadori et al (2012), because the weight counter is represented using a finite number of bits l, the method is only able to handle a subset of the learning rate, that satisfies this condition:

$$\frac{s(w_0, \alpha_{min}^{w})}{w_0} \leq \frac{L}{STEP} \tag{41}$$

Consequently, using Eq. 34, the retrieved interval of learning rate is:

$$\alpha \geq 1 - (1 - w_0)^{\frac{STEP}{w_0 \cdot L}} \tag{42}$$

To increase the range of this interval, the stochastic updating method described in Sec. 3.5.1 is applied. Since we are using a linear approximation, the probability $P_{STEP}(n)$ to update the weight (counter) is constant and independent to n (it depends only on the learning rate α). Consequently, it is possible to compute its value solving Eq. 37:

$$P_{STEP}(\alpha) = \frac{L}{STEP \cdot A} \tag{43}$$

Thus, in this case $P_{STEP}(n)$ represents the ratio between the maximum value $\frac{L}{STEP}$ that can be represented in l bits (namely $L = 2^l - 1$), and the number of steps A needed to reach the value $w = 1$ using Eq. 40.

3.5.3 iGMM-s: Staircase approximation

This technique is based on the assumption that all the possible weight values may be represented by an exponential distribution as defined by Eq. 32 or Eq. 33 that has been uniformly sampled over the iterations. For illustrative purposes, only the increasing function is considered in a first instance, being the decreasing function symmetrical as described later. Such approach emulates completely the weight trend of Eq. 32, but the above

counter may require more levels than what is available in the system. Therefore, we propose an approximated solution. The increasing exponential curve in Fig. 4 can be calculated as a linear function sampled using an exponentially distributed sampling period over the iterations (see dotted line in Fig 5b).

$$\bar{\nu}(s,\alpha) = w(s) - w(s-1) = (1-\alpha)^{(s-1)}\alpha \tag{44}$$

Following this logic, the weight is a linear function that can be represented as an integer counter c .

$$w(c) = c \cdot \nu \tag{45}$$

Thus, assuming that the available number of bits to represent the weight is l, and consequently the number of available levels to represent the weight are $L = 2^l - 1$ (0 represents the value $w = 0$), the idea is to sample the interval $[0, 1]$ using at least L points. To optimise both memory footprint and processing complexity, the learning rate (namely the maximum value in Eq. 44 when $s = 1$) is chosen as sampling period ν, and consequently the counter c is updated stochastically, in order to approximate the trend of Eq. 32. Therefore, the stochastic updating method of Sec. 3.5.1 is the core mechanism used for updating the weight counter for iGMM-s: the weight is updated with a exponential distributed probability dependent on the value of the counter c ($P_{STEP}(c)$). Because the chosen sampling period is the maximum inside the set described in Eq. 44, this probability is defined as how many iterations are compressed at every counter updating of the value $STEP$:

$$P_{STEP}(c,\alpha) = \frac{STEP}{\bar{\nu}(c,\alpha)} \tag{46}$$

Thus, a *Look-Up Table* (or LUT) is instantiated inside a vector of L elements to contain the thresholds computed as a function of $P_{STEP}(c,\alpha)$ using Eq. 39: we call this data structure *threshold-LUT* or $LUT_T(\alpha)$. Therefore, from this vector indexed by a counter c (namely $LUT_T(\alpha)[c]$), it is possible to obtain the current threshold value for the stochastic updating technique (see also Sec. 3.5.1).

All these considerations only refer to the weight increasing operations and they do not take into account the weight decreasing case. As seen in Fig. 4 the increasing trend curve is symmetric to the decreasing trend curve with respect to the horizontal line through 0.5. Therefore, it is simple to conclude that the *updating step function* has the same form in both the increasing and the decreasing trend, with the difference that the first one maps the update from 0 to 1, the second one the inverse, namely from 1 to 0. It is then possible to state that the same *threshold LUT* can be used for both the increasing and the decreasing trend by simply inverting the access index. Summarizing:

1. if the pixel p_i belongs to the current Gaussian, the counter c is increased by $STEP$ if and only if the random value $\mathbf{V} > LUT_T(\alpha)[c]$;
2. if the pixel p_i does not belong to the current Gaussian, the counter c is decreased by $STEP$ if and only if the random value $\mathbf{V} > LUT_T(\alpha)[\bar{S} - c]$, where $\bar{S}$ is the number of iterations needed to reach the weight value 0.99 from 0;
3. otherwise no operation is performed.

4 Implementation

This section discusses the implementation details of the proposed methods. Firstly, the hardware specifications of two platforms are presented: the *SEED-EYE* board (Scuola Superiore Sant'Anna and Evidence s.r.l. (2011)) and the *Raspberry Pi* board (Raspberry Pi Foundation (2011)). Then, the memory and processing requirements of our methods, deployed on the above platforms, are discussed.

4.1 The selected boards

4.1.1 The SEED-EYE board

The *SEED-EYE* board is an advanced multimedia Smart Camera Networks node based on PIC32MX795F512L by Microchip (see also Microchip (2008)) that embeds: a wireless transceiver compliant with the IEEE802.15.4 standard, an IEEE802.3 (Ethernet) interface, a USB interface and a CMOS camera. Low power operation is an essential requirement for such mobile sensors and the specific hardware specifications allow the board operating at 0.45W with only 0.30W consumed by the microcontroller.

The PIC32 microcontroller is characterized by a computational capability of 80MHz, and an internal RAM of 128KBytes. It lacks FPU and consequently any floating point data are handled by software libraries. In Tab. 3 the processing time for executing a series of 32 millions arithmetic operations (+, -, *, /) on both floating point and integer data is shown: for every arithmetic operation, the ratio between the processing times of on floating point data (namely *float*) and on integer data T_{double}/T_{uint32_t} is at least 11 (see Tab. 3). This ratio hints at the potential gain of our integer-based implementation.

4.1.2 The Raspberry Pi board

The Raspberry Pi is based on a Broadcom BCM2835 system on a chip (SoC), which includes an ARM1176JZF-S characterized by a computational capability of 700 MHz processor and 256 MBytes of RAM. An SD card is used

Table 3: Standard operation times using integer and floating point representation for Raspberry Pi and SEED-EYE boards

Precision	Sum [s]		Subtraction [s]		Multiplication [s]		Division [s]	
	SEED-EYE	Rasp. Pi	SEED-EYE	Rasp. Pi	SEED-EYE	Rasp. Pi	SEED-EYE	Rasp. Pi
uint32_t	3.31	0.29	3.43	0.30	4.11	0.47	9.80	0.76
double	41.98	0.86	50.08	0.85	44.97	0.91	130.58	2.01
Ratio T_{double}/T_{uint32_t}	∼13	∼3	∼15	∼3	∼11	∼2	∼13	∼2.5

for booting and long-term storage. Although this system lacks FPU, floating point arithmetics are supported by an optimised library which is part of the operation system *Raspbian* (Debian project (2012)): integer operations are about 2-3 times faster than floating point operations (see Tab. 3), therefore an integer-based implementation may still be valuable.

Table 4: Algorithm footprint (QQ-VGA images)

Precision	Bytes per Gaussian	Footprint 2G (Bytes)	Footprint 3G (Bytes)
Double	24	921,600	1,382,400
Float	12	460,800	691,200
uint8_t	3	115,200	172,800
iGMM-x	2	76,800	115,200

4.2 Algorithm optimizations

Usually microcontroller/microprocessors for embedded systems are characterized by both limited memory and computational capability to reduce the size of the boards, the energy consumption and the cost. Therefore, our application code has been appropriately optimised using standard techniques (i.e., *loop-unroll* and *word-unroll*), to further reduce the computational cost. In addition, the implementation of the stochastic approach (section) is achieved by a pseudo-random number generator Elia and Neri (1986), which is used only once per frame: the same pseudo-random value is used to update all pixel weights of the same frame.

All the missing implementation details are available by request from the authors.

4.2.1 GMM representation

In this section the memory footprint of the data structure used to store the background model is discussed. In Tab. 4 the memory footprint of the GMM background model for different data types is shown. Using a standard integer type (*uint8_t*) memory usage is reduced by a factor 4 and 8, compared to floating point types such as (*float*) and (*double*) respectively. to permit the deployment of the GMM algorithms over the above mentioned memory constrained architectures, we propose to represent each Gaussian in a compressed version, namely using only 2 bytes, distributed between its three parameters.

Thus, the mean value, usually defined in the range $[0, 255]$, is represented using 7 bits and with a granularity $\gamma_\mu = 2$, such that only the even numbers are considered. Such a choice is derived from the noise introduced by the hardware components, characterized by a standard deviation of $2 - 2.5$ (Hwang et al (2007)): a granularity smaller than the noise is pointless.

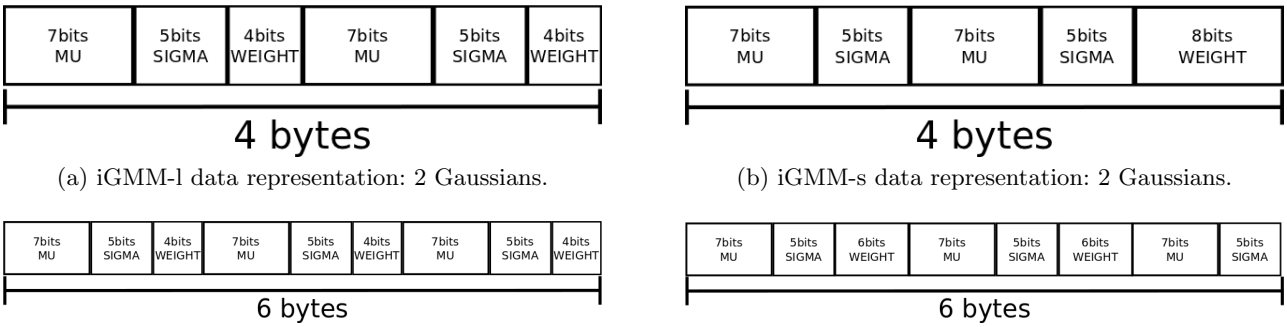
For the same reason,we propose the interval $[5, 36]$ that represents a reasonable set of variance values with respect to the background pixels distribution acquired with real cameras under no changing light conditions. Therefore, the variance is sufficiently represented using 5 bits and with a granularity of 1. Finally, the weight has two different representations depending on the chosen proposed approach. iGMM-l associates a weight (represented using 4 bits) to each Gaussian as in the original GMM. iGMM-s instantiate $G-1$ weights represented using a minimum of 6 bits, where the last G-th weight is retrieved as a function of the previous $G-1$, using Eq. 7. Fig. 6a and Fig. 6c shows the bit distribution for the two and three Gaussians mixture examples.

4.2.2 Learning rate operating range

The usage of an integer based representation for native floating point based data introduces a limitation on the learning rate values due to Eq. 9. Therefore, our method operates properly only for a range of learning rate values, which we call *learning rate operating range*. This range is specified as the set of value of α such that:

$$\alpha \geq \frac{1}{P} \tag{47}$$

In the proposed implementation P is fixed to 10,000, and consequently the *learning rate operative range* is the interval $[0.0001, 1]$, which is sufficient to cover almost all practical cases and it is much wider than the one proposed in Salvadori et al (2012). Particularly, in Tab. 5 a comparative table is shown to demonstrate the enhancements of the learning rate operating range derived from



(a) iGMM-l data representation: 2 Gaussians.

(b) iGMM-s data representation: 2 Gaussians.

(c) iGMM-l data representation: 3 Gaussians.

(d) iGMM-s data representation: 3 Gaussians.

Fig. 6: iGMM-x data representation.

the proposed work with respect to our previous work in Salvadori et al (2012).

Table 5: Learning rate operating range: comparison with Salvadori et al (2012)

	α_{min}	α_{MAX}
Salvadori et al (2012)	> 0.125	≤ 1
Proposed solution	0.0001	≤ 1

5 Performance evaluation

In this section the performance of the proposed algorithm is shown. Particularly, using a simulation approach, the three Gaussian parameter trends are validated to satisfy the claims described in Sec. 3.3 and 3.5. Additionally, the performance of our approach is evaluated both qualitatively and quantitatively using two standard data-sets which comprise different kind of movement: the "IXMAS data-set" [Weinland et al (2006)] (slow movement) and the "Fudan Pedestrian data-set"
[Ben Tan (2011)] (fast movement). Finally the processing time of our implementations (i.e., iGMM-l and iGMM-s) is compared with an optimized version of the original GMM over the architectures described on Sec. 4.1.

5.1 Validation of the parameters

5.1.1 μ and σ^2 updating calibration and validation

Setting the iGMM-x technique requires only the calibration of the values N_μ and N_{σ^2} as discussed in Sec. 3.3. This operation maximises the overlap between the membership areas of the original GMM and the proposed iGMM-x under different illumination conditions, by varying N_μ and N_{σ^2} on their domain (namely the interval $(0, 1)$).

We simulate three different illumination conditions: (i) variable illumination (i.e. day/night transition) with a rate of 8.68 levels per minute, (ii) variable illumination (i.e. cloud/sun transition) with a rate of 4.34 levels per minute, and (iii) constant illumination. Specifically, the illumination trend is approximated as a linear transition, and the typical noise introduced by the acquisition systems is emulated using a Gaussian number generator. In this process, the iGMM-x membership area is filtered using a moving window proportional to $1/\alpha$, to reduce its fast oscillations.

In the case of uncalibrated iGMM-x, simulation results in Fig. 9 and 10 reveal that the iGMM-x membership area (red) differs significantly from the GMM one (green), when average illumination is varying. On the other hand, when average illumination is constant, lack of calibration does not influence the membership areas (Fig. 14 and Fig. 11) as pixel matching is straightforward.

Fig. 7 and 8 depict the optimal values of N_μ and N_{σ^2} respectively that maximise the overlap between the iGMM-x membership area with the GMM one (red lines): we approximate those trends by the blue lines to correctly tune the updating step ξ (see Sec. 3.4).

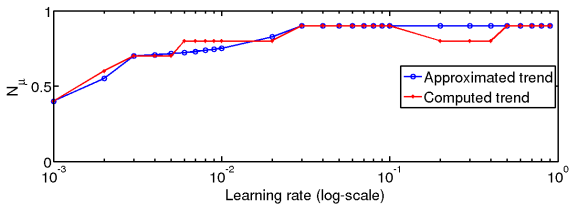
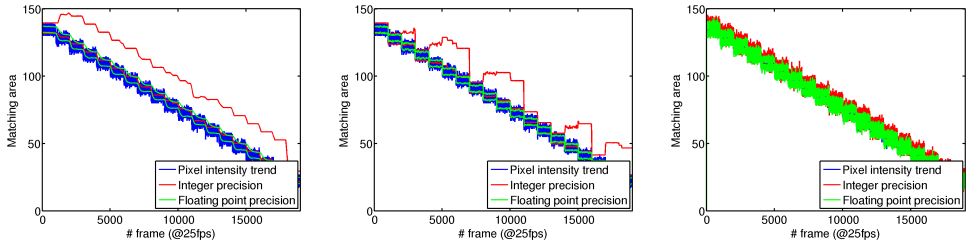


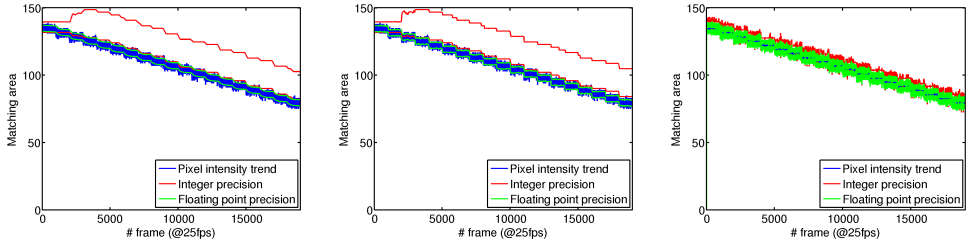
Fig. 7: N_μ trend.

In Fig. 12 and 13 the membership area of the calibrated iGMM-x approach (red) is compared with the original GMM (green) under two different rates of illumination transitions and different values of learning rate.



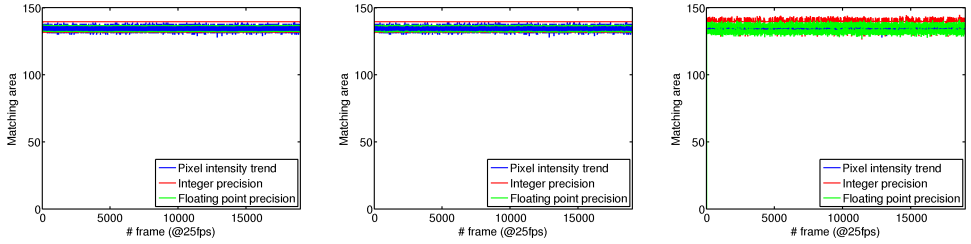
(a) $\alpha = 0.005$ (b) $\alpha = 0.05$ (c) $\alpha = 0.5$

Fig. 9: Variable illumination @8.68 levels per minute: un-calibrated case.



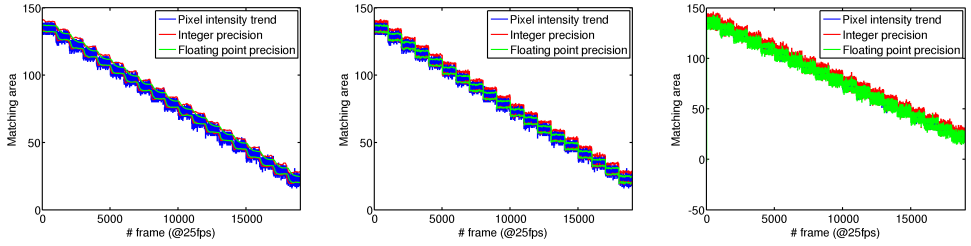
(a) $\alpha = 0.005$ (b) $\alpha = 0.05$ (c) $\alpha = 0.5$

Fig. 10: Variable illumination @4.64 levels per minute: un-calibrated case.



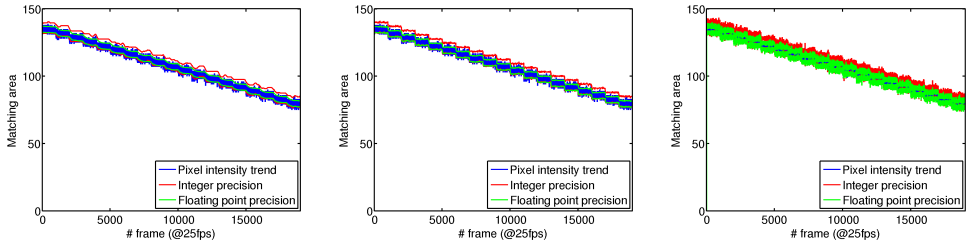
(a) $\alpha = 0.005$ (b) $\alpha = 0.05$ (c) $\alpha = 0.5$

Fig. 11: Constant illumination: un-calibrated case.



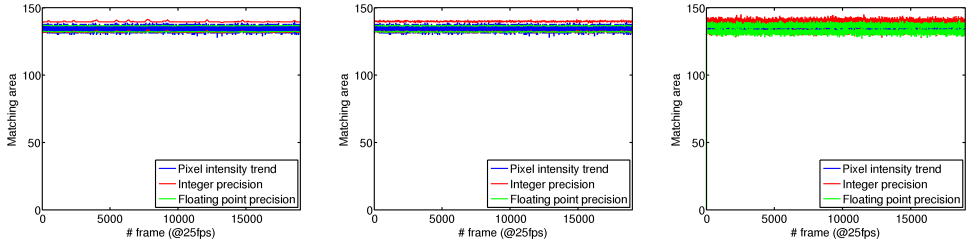
(a) $\alpha = 0.005$ (b) $\alpha = 0.05$ (c) $\alpha = 0.5$

Fig. 12: Variable illumination @8.68 levels per minute: calibrated case.



(a) $\alpha = 0.005$ (b) $\alpha = 0.05$ (c) $\alpha = 0.5$

Fig. 13: Variable illumination @4.34 levels per minute: calibrated case.



(a) $\alpha = 0.005$ (b) $\alpha = 0.05$ (c) $\alpha = 0.5$

Fig. 14: Constant illumination: calibrated case.

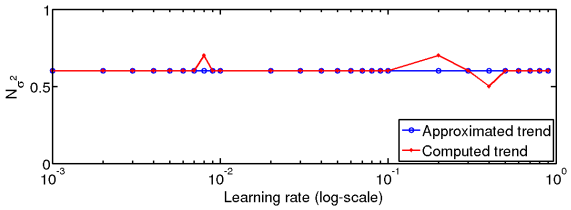


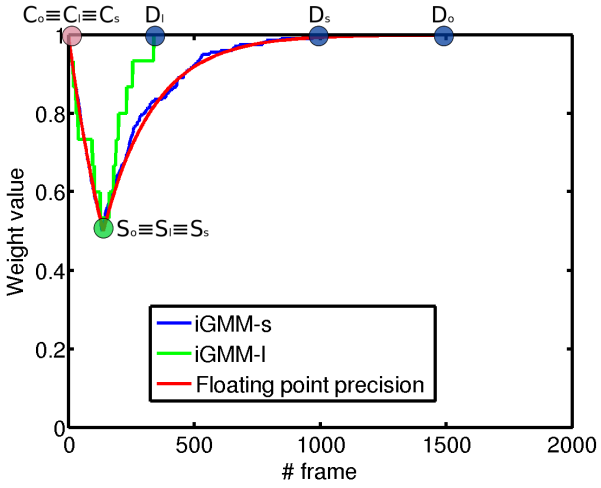
Fig. 8: N_σ^2 trend.

5.1.2 Weight trend validation

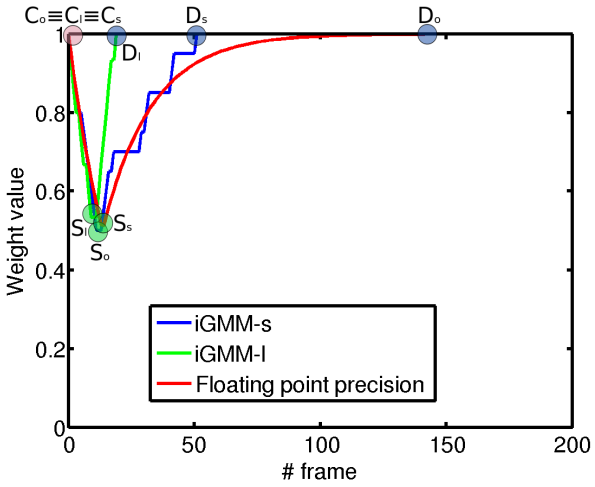
In Sec. 3.5, two possible approximations are defined and stochastic updating techniques (see also Sec. 3.5.1) are used in both of them. To validate the proposed approximations, we propose a simulative approach of the lifetime of Gaussians. As a starting condition, we consider a mixture where only the background mode MG_1 is active, and the effect of the creation and the evolution on a new Gaussian MG_2 is evaluated to understand how the MG_1 weight trend develops. Particularly, we propose to compare the evolution of the approximated techniques with the original GMM weight trend, especially taking into account the occurrence of the following three points: (i) the C point where the MG_2 mode is created (*creation point*), (ii) the S point, where MG_2 mode is sorted as most popular Gaussian (*swap point*), and (iii) the D point where the MG_1 is destroyed (*destruction point*). Thus, both iGMM-x techniques emulate correctly the creation point (see the C_l and C_s points in in Fig. 15). Moreover, the iGMM-l approximates linearly the weight trend (the green line in Fig. 15) and emulates correctly the Gaussian-swap point (see the point S_l), but it discards Gaussians too early (see the point D_l) in comparison with the original GMM case (the red line in Fig. 15). On the other hand, iGMM-s approximates as a staircase the weight trend (the green line in Fig. 15) and emulates correctly both the swap and the destruction point (see both the points S_s and D_s) compared with the GMM case. Consequently it is possible to state that, the iGMM-s technique approximates better the weight trend of the original approach. However, for large values of the learning rate, both cases have equivalent trends because the threshold T_X (see Sec. 3.5.1) for the staircase approximation is a very small number (see Fig 15c).

5.2 Comparison to the original GMM algorithm

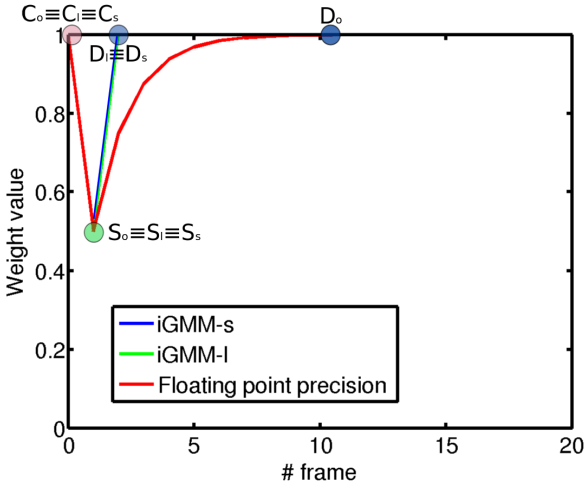
In this section the performance for both iGMM-l and iGMM-s is compared to standard GMM by means of a qualitative analysis, i.e., a visual comparison on binarized foreground images, and quantitative analysis, i.e., an analysis on aggregated metrics. Finally in Sec. 5.2.2 the processing time of the proposed optimization (Sec. 3) over both the Raspberry Pi and the SEED-EYE boards is measured for varying learning rate.



(a) $\alpha = 0.005$



(b) $\alpha = 0.05$



(c) $\alpha = 0.5$

Fig. 15: Weight trends, where C_o, C_l and C_s are the creation points, S_o, S_l and S_s are the swap points, and D_o, D_l and D_s are the destruction points for GMM, iGMM-l and iGMM-s respectively.

5.2.1 Qualitative and quantitative comparison

Firstly, a qualitative comparison between the binarized foreground images generated by the iGMM-x, the standard GMM and the ground-truth is shown (Tab. 6 and 7) for both data-sets: our approaches have comparable results to the original GMM method in foreground segmentation.

To understand the overall performance of the proposed algorithms, an aggregate analysis, based on Precision-Recall (or P-R) curves is presented. Particularly, the minimum distance of the P-R curves from the perfect classification point is computed for each value of learning rate so that the smaller the distance, the better the performance. In Fig. 16, the trends of the distance for P-R curves are shown as a function of the learning rate for both data-sets and all the cases (i.e., mixture of 2 and 3 Gaussians).

In Fig. 16c and 16d the evaluation on the fast movement data-set (namely Fudan data-set) are shown: in this case the considered metric has a constant trend because only few objects are absorbed by the background. On the other side, in Fig. 16a and 16b the impact of the considered techniques on the slow movement data-set (namely using IXMAS data-set) is shown. In this case the segmentation performance decrease with the increase of the learning rate, due to the foreground objects absorption inside the background. Apart the different trends, all the results shown in these figures demonstrate that the iGMM-x has comparable performance with original GMM.

5.2.2 Processing time comparison

In this section, the processing time of iGMM-l, iGMM-s and GMM techniques are compared to measure the real impact of implementations over the two boards described on Sec. 4.1. Specifically, on the Raspberry Pi board two different resolutions are tested (Q-VGA and QQ-VGA) and on the SEED-EYE board only the 40x40 resolution is used in order to fit the scarce amount of PIC32 RAM for configurations of 2 and 3 Gaussians.

In Tab. 8, the mean values of the processing times are shown for all the different configurations in terms of used platform, resolutions and number of Gaussians. iGMM-l is 3÷5 times faster than GMM and iGMM-s 1.5÷3 times on the Raspberry Pi board.These ratios increase by a factor 10 (iGMM-l is around 40 times faster than GMM and iGMM-s 20÷28 times) on the SEED-EYE board. The obtained results are consistent with the data shown in Tab. 3 in Sec. 4. Processing time of iGMM-s is longer than the one of iGMM-l due to the higher complexity of the staircase approximation (e.g., iGMM-s needs the use of "read" operations from the LUT_T buffer for every pixel). Moreover comparing the experiments with 2 and 3 Gaussians (in the same platform and with a constant resolution), the complexity of iGMM-s increases

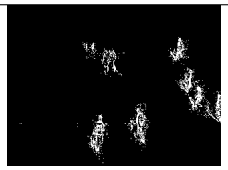
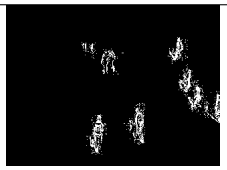
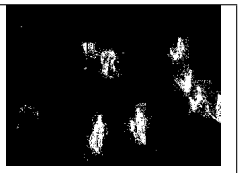
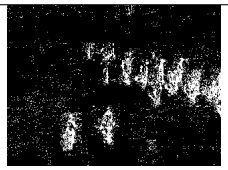
Picture	Ground-truth	GMM	iGMM-l	iGMM-s
				
				

Table 6: "Fudan pedestrian" data-set results using a 3 Gaussians mixture.

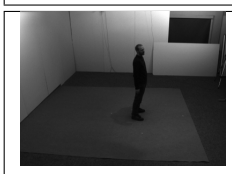
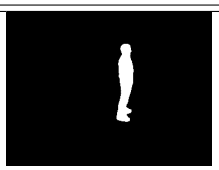
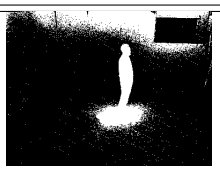
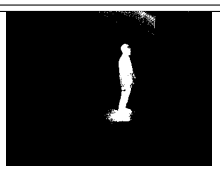
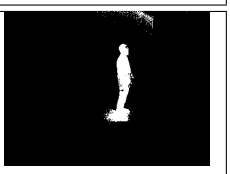
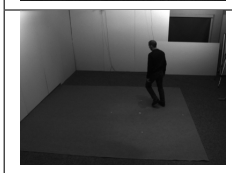
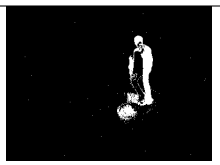
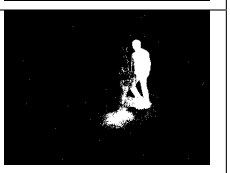
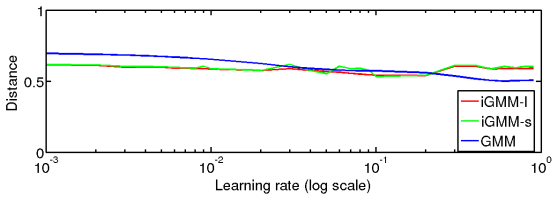
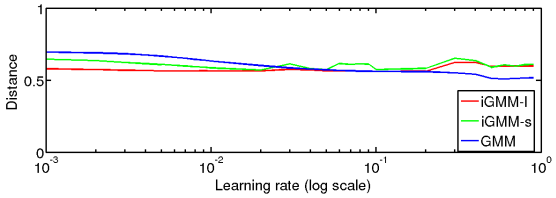
Picture	Ground-truth	GMM	iGMM-l	iGMM-s
				
				

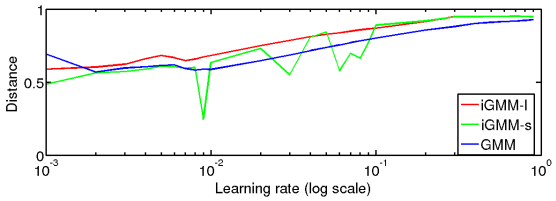
Table 7: "IXMAS" data-set results using a 3 Gaussians mixture.



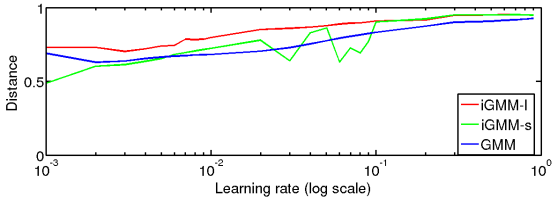
(a) Using "Fudan pedestrian" data-set (2 Gaussians).



(b) Using "Fudan pedestrian" data-set (3 Gaussians).



(c) Using "IXMAS" data-set (2 Gaussians).



(d) Using "IXMAS" data-set (3 Gaussians).

Fig. 16: Minimum distance from the P-R curves to the perfect classification (1,1)

Table 8: Processing times

	GMM [s]	iGMM-l [s]	iGMM-s [s]
Rasp.Pi 2G QQ-VGA	0.033	0.007	0.010
Rasp. Pi 3G QQ-VGA	0.036	0.013	0.025
Rasp. Pi 2G Q-VGA	0.136	0.031	0.041
Rasp. Pi 3G Q-VGA	0.148	0.057	0.104
SEED EYE 2G (40x40)	0.084	0.002	0.003
SEED EYE 3G (40x40)	0.115	0.003	0.006

significantly with the number of Gaussians with respect to iGMM-l: the latter is about 1.5 times faster than the former in case of two Gaussians mixture, but about 2 times in case of three Gaussians.

6 Conclusions

In this paper we have proposed two sub-integer precision realisations of mixture of Gaussian modelling to fit the strict constraints of embedded microcontrollers with the lack of FPU. To this purpose, we have redefined the updating rules of each Gaussian parameter. Specifically, the mean value and the variance of each Gaussian are updated by a redefined and generalised "round" operation that emulates the original updating rules for a large set of learning rates. On the other hand, weights are represented by counters that are updated following stochastic rules to allow a wider range of learning rates and the weight trend is approximated by a line (iGMM-l) or a staircase (iGMM-s).

Experimental results show that both integer realisations have comparable accuracy on background/foreground segmentation compared to the original floating point precision, but significantly smaller memory footprint and lower computational cost over both the considered hardware platforms. Specifically, in our implementations, memory requirements are 6 and 12 times lower than *float* and *double* precision versions respectively. Processing time is reduced by 30%-79% on the Raspberry Pi board and by 95%-98% on the SEED-EYE board. Such a difference is justified by the usage of an optimised floating point library for the implementation of the original GMM on the Raspberry Pi board. The two proposed versions, iGMM-l and iGMM-s have similar performance and memory footprint. However, iGMM-l is faster by 50%-100% than iGMM-s, and the higher the number of Gaussians the higher the difference, due to the complexity of handling the staircase weight-trend.

Appendices

A Proof from Sec. 3.5.1

A uniformly distributed random number $\mathbf{X} \in [0, MAX]$ is characterised by the cumulative distribution function in Eq. 48 and the probability density function in Eq. 49.

$$F_X(x) = P(\mathbf{X} \leq x) = \frac{x}{MAX} \tag{48}$$

$$f_X(x) = \frac{\partial}{\partial x} F_X(x) = 1 \tag{49}$$

On the other hand, if we consider the value $P(\mathbf{X}_n > T_X)$ and we use the probability properties, the following relation can be obtained:

$$P(\mathbf{X} > T_X) = 1 - F_X(x) = 1 - \frac{T_X}{MAX} \tag{50}$$

From the assumption in Eq. 38, Eq. 39 is true.

■

References

Apewokin S, Valentine B, Forsthoefel D, Wills L, Wills S, Gentile A (2009) Embedded real-time surveillance using multimodal mean background modeling. In: Kisaanin B, Bhattacharyya SS, Chai S (eds) Embedded Computer Vision, Advances in Pattern Recognition, Springer London, pp 163–175

Ben Tan LW Junping Zhang (2011) Semi-supervised elastic net for pedestrian counting. Pattern Recognition

Candes E, Romberg J, Tao T (2006) Robust uncertainty principles: exact signal reconstruction from highly incomplete frequency information. Information Theory, IEEE Transactions on 52(2):489 – 509, DOI 10.1109/TIT.2005.862083

Cheung SCS, Kamath C (2004) Robust techniques for background subtraction in urban traffic video. SPIE, vol 5308, pp 881–892, DOI 10.1117/12.526886

Cucchiara R, Grana C, Piccardi M, Prati A (2003) Detecting moving objects, ghosts, and shadows in video streams. Pattern Analysis and Machine Intelligence, IEEE Transactions on 25(10):1337 – 1342, DOI 10.1109/TPAMI.2003.1233909

Debian project (2012) Raspbian. http://www.raspbian.org/

Donoho D (2006) Compressed sensing. Information Theory, IEEE Transactions on 52(4):1289 –1306, DOI 10.1109/TIT.2006.871582

Elia M, Neri F (1986) Generation of pseudorandom independent sequences. In: IASTED International Symposium MIC '86

Hung MH, Pan JS, Hsieh CH (2010) Speed up temporal median filter for background subtraction. In: Proceedings of the 2010 First International Conference on Pervasive Computing, Signal Processing and Applications, IEEE Computer Society, Washington, DC, USA, PCSPA '10, pp 297–300, DOI 10.1109/PCSPA.2010.79, URL http://dx.doi.org/10.1109/PCSPA.2010.79

Hwang Y, Kim JS, Kweon IS (2007) Sensor noise modeling using the skellam distribution: Application to the color edge detection. In: Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on, pp 1–8, DOI 10.1109/CVPR.2007.383004

Iannizzotto G, La Rosa F, Lo Bello L (2010) A wireless sensor network for distributed autonomous traffc monitoring. In: Human System Interactions (HSI), 2010 3rd Conference on, pp 612 –619, DOI 10.1109/HSI.2010.5514504

Lo B, Velastin S (2001) Automatic congestion detection system for underground platforms. In: Intelligent Multimedia, Video and Speech Processing, 2001. Proceedings of 2001 International Symposium on, pp 158 –161, DOI 10.1109/ISIMP.2001.925356

McFarlane NJB, Schofield CP (1995) Segmentation and tracking of piglets in images. Machine Vision and Applications 8(3):187–193, DOI 10.1007/BF01215814

Microchip (2008) PIC32MX3XX/4XX Family Data Sheet. ww1.microchip.com/downloads/en/DeviceDoc/61143E.pdf

Pagano P, Salvadori C, Madeo S, Petracca M, Bocchino S, Alessandrelli D, Azzar A, Ghibaudi M, Pellerano G, Pelliccia R (2012) A middleware of things for supporting distributed vision applications. In: Proceedings of the 1st Workshop on Smart Cameras for Robotic Applications (SCaBot)

Piccardi M (2004) Background subtraction techniques: a review. In: Systems, Man and Cybernetics, 2004 IEEE International Conference on, vol 4, pp 3099 – 3104 vol.4, DOI 10.1109/ICSMC.2004.1400815

Radke RJ, Andra S, Al-Kofahi O, Roysam B (2005) Image change detection algorithms: A systematic survey. IEEE Transactions on Image Processing 14:294–307

Rahimi M, Baer R, Iroezi OI, Garcia JC, Warrior J, Estrin D, Srivastava M (2005) Cyclops: In situ image sensing and interpretation in wireless sensor networks. In: In SenSys, ACM Press, pp 192–204

Raspberry Pi Foundation (2011) Raspberry Pi BOARD. http://www.raspberrypi.org/

Salvadori C, Makris D, Petracca M, del Rincón JM, Velastin SA (2012) Gaussian mixture background modelling optimisation for micro-controllers. In: ISVC (1), pp 241–251

Scuola Superiore Sant'Anna and Evidence srl (2011) SEED-EYE BOARD. http://www.evidence.eu.com/products/seed-eye.html

Shen Y, Hu W, Liu J, Yang M, Wei B, Chou C (2012) Efficient background subtraction for real-time tracking in embedded camera networks. In: Proceedings of the 10th ACM conference on Embedded Network Sensor Systems

Stauffer C, Grimson WEL (1999) Adaptive background mixture models for real-time tracking. In: Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on., vol 2, DOI 10.1109/CVPR.1999.784637

Weinland D, Ronfard R, Boyer E (2006) Free viewpoint action recognition using motion history volumes. Computer Vision and Image Understanding 104(2-3):249–257